Task 1: String Handling (0 points)

Write a program that asks the user to enter 2 numbers, each consisting of a maximum of 4 digits. The program shall verify that the numbers are in the correct format, and then calculate and print the sum of them.

The numbers should be read as C-strings, i.e., array of char. If any digit string exceeds the specified limit, or if any character in each string is a letter, an appropriate error message should be printed and the program should finish. Otherwise, the strings are converted to integers to finally be summarized. It may be appropriate to also verify that any characters actually are input, i.e., user does not press *Enter* key directly without typing. At least two standard functions shall be used.

Although there is a new string type in C++ that simplify string handling, it is in many contexts essential to master the original C style, such as char str[10], which is the data type we call ***C-strings***. In the standard libraries for C is a collection of functions for the handling of characters and strings. You should be familiar with these, in order to solve the program above. Note that there are many options for *error handling* in connection with input of data. In this task, one possible way is shown, slightly adjusted in order to facilitate the understanding of the structure and function of strings.

Test case

Run example

```
Enter two numbers (maximum 4 digits).

Number 1: 12          (Ok)
Number 2: 12345       (Too many numbers)
Try again!
Number 2:             (Too few numbers)
Try again!
Number 2: 34x4        (Includes letters)
Try again!
Number 2: 3456        (Ok)

The sum is 3468
```

Tips

Think about how a string is represented internally, and how you can access the values for each one of them to verify them. Note that ”numbers” you enter via the keyboard are interpreted differently; depending on what data type they are assigned to. If an integer is expected (e.g. an int), the number 123 is read as a numerical value. If a string is expected, the three characters ’1’, ’2’ and ’3’ is loaded to an array, ended with the character ’\0’.

Solution

A C string is represented by a char array (an array with characters), which always is null terminated with the null termination character for strings: '\0'. This makes it easy to control or manipulate individual characters, which are accessed using indexing. With other words, you could read each *number* to a string, which is then examined character by character in a loop.

Source code

```
//
// File:    StringHandling.cpp
// Summary: This program reads two strings from user, to be seen
// as numeric values. The program performs format checking, then
// converts the strings to integer values, before it calculates
// and displays the resulting sum.
// Version: 1.1


#include <iostream>
#include <cstdlib>   // atoi()
#include <cctype>    // isdigit()

using namespace std;

// Prototypes
bool stringOk(char s[], int amount);

const int MAXLEN = 100;    // Max length of temp strings
const int AMOUNT = 4;      // Max amount of digits

int main()
{
   // String for reading the number - always use constant for the size
   char str[MAXLEN] = {'\0'};

   int number1 = 0, number2 = 0;
   bool ok;

   cout << "Enter two numbers (max 4-digits).\n\n";

   do
   {
      cout << "Number 1: ";
      cin.getline(str, MAXLEN, '\n');
      // Reads max 99 characters, adds '\0' in the end.
      // '\n' means that reading is cancelled at the end-of-line character ('\n' is default),
      // which you get by pressing Enter - '\n' is finally removed from the input stream.
      ok = stringOk(str, AMOUNT);

      if (!ok)
         cout << "Try again!" << endl;
   } while (!ok);

   number1 = atoi(str); // atoi() returns the numerical value of the string


   do
   {
      cout << "Number 2 ";
      cin.getline(str, MAXLEN);
      // The third argument to getline can be omitted - default it is '\n'
      ok = stringOk(str, AMOUNT);

      if (!ok)
         cout << "Try again!" << endl;
   } while (!ok);

   number2 = atoi(str);

   cout << "\nThe sum is " << number1 + number2 << endl;

   return 0;
}
```

```
//
// bool stringOK(char s[], int amount);
// Returns true if s contains of at least 1 digit and max amount digits, else false
//
bool stringOk(char s[], int amount)
{
   int i = 0;

   while (s[i]!= '\0')        // Is often written: while (s[i])
   {                          // that is, as long as s[i] has a value !=0
      if (!isdigit(s[i]))     // if no digit
         return false;        // cancel
      ++i;
   }

   return (i > 0 && i <= amount);   // true if we have found 1 more digit
}
```

What can you learn from this program?

- Whenever you define strings and arrays, you should always specify desired length. You usually have to make sure that this length is not exceeded at various points in the program. Therefore, you should ALWAYS define the length as constant, to make sure you use the same value in all places in the program.
- **cin.getline()** is a more modern way of reading than cin.gets(). This function should be used for string reading, because it (in opposite to **cin**) reads *whitespaces* (”unwriteable” characters such as spaces, tabs, line breaks) and also adjusts the number of read characters to the specified string length. It is even possible to specify a specific *delimiter* (or ”separator”), that is, the character at which the reading should be stopped. The delimiting character (which as *default* is ’\n’) is then thrown.

  The function of cin.getline() has been changed and can work different if you use older compilers. This is about how failbit flag is set for empty strings and the effect in cases of too long strings.

  If you are facing any problems, due to an older compiler, you can instead test cin.get().
- Strings should always be **reset to zero**, in order to avoid difficult-to-detect, logic errors (on which compiler will not react). Use one of the methods suggested above. The initialization statement `char str[MAXLEN] = {'\0'};` guarantees that ENTIRE string is reset, while an assignment of an “empty string” (`char str[MAXLEN] = "";`) will do it *in most cases* (compiler dependent?). Standard functions that are permitted to operate on strings, although not null-terminated, can give misleading results.
- A Boolean variable (**bool ok** in this example) is often useful, in order to verify ”error status” of an application. We do not have to call stringOk more than once per entry, as we need to check twice if the string is OK.
- Old standard functions for characters and strings often return the data type *int* with the values 1 or 0, which can be interpreted as **true** or **false**. It can be used directly ”inside” an **if**- or **while**-condition, to test if the function returns true or false.

  Example:

  ```
  if (!isdigit(s[i]))     // if no digit
        break;            // cancel
  ```
- Because two strings are to be tested with the essentially same conditions, the tests have been stored in an own function: `bool stringOk(char s[], int amount)` which step by step counts the number of characters in the string. If it’s not a digit, the step-by-step counting is cancelled and returns `(i > 0 && i <= amount);` which

is true if it is 1-4 digits, otherwise false.

- Notice that the string *str*, which is used for the input, also can be reused to read input for number2.

Task 2: Diving Score (3 points)

In the sport of diving, each dive taken by a diver has a degree of difficulty which ranges from 1.0 to 4.0. For each dive, *seven* judges award a score between 0 and 10, where each score may be a floating-point value.

After all judges have issued scores, a diver's total score is computed as follows:

- Out of the seven judges' scores, excluding the highest and lowest scores and take a sum of the remaining five score. If two or more judges give same higher or lower score, only one instance of the higher or the lower score is excluded, see the example run below.
- Multiply the sum of the five judges scores with the degree of difficulty of the dive.
- The resulting total of multiplication from the above step is then multiplied by 0.6

Write a computer program that inputs a degree of difficulty and seven judges' scores, and outputs the overall score for that dive as shown in the following example run. The program is required to use an array of appropriate type and size to store and process the judges' score. The program should ensure that all inputs are within the allowed data ranges. If a user enters an out of the range value, for instance, score less than 0 or more than 10, the program notifies the invalid value to the user and ask to enter the value again.

Example Run

```
Enter the degree of difficulty for the dive (1.0 - 4.0): 1.3
Enter score for judge 1 (0-10): 8
Enter score for judge 2 (0-10): 6
Enter score for judge 3 (0-10): 7
Enter score for judge 4 (0-10): 4
Enter score for judge 5 (0-10): 3
Enter score for judge 6 (0-10): 5
Enter score for judge 7 (0-10): 4

The diver's final score is 20.28
```

Presentation

The task is to be solved as follows:

- A fully working program that meets the specified criteria.
- A program code that is well-written and easy to read, according to the coding standards published on the course home page.

Task 3: Reducing Fractions to Lowest Terms (3 points)

Issue

Write a program that asks the user to enter the values of a fraction. A structure, with data members that can represent the *numerator* and the *denominator,* shall be used to store the fraction. Moreover, the program should satisfy the following requirements.

1. The program should reduce the fraction to its lowest terms and print the lowest terms as shown in the example output displayed below as a test case.
2. After reducing the fraction to lowest terms, the program should represent (print) the fraction in mixed form, that is, for example, 3/2 = 1½, or 25/7 = 3 4/7
3. It is not common to use a negative denominator (eg: 4/-3). In such cases, the fraction should be adjusted so that denominator instead becomes positive and the numerator switches sign (eg: -4/3).
4. Multiple calculations should be able to perform without restarting the application.

Test case

Run example

```
FRACTION CALCULATION
===========

Enter the numerator: 125
Enter the denominator: 35

The fraction can be abbreviated to : 25/7 = 3 4/7
```

Tips

To reduce the fraction to lowest terms, first bring out the *gcd*, greatest common divisor, for the fraction.

Presentation

The task has to be solved as follows:

- A fully working program that meets the specified criteria.
- A program code that is well-written and easy to read, according to the coding standards published on the course home page.

Task 4: Man or Woman? (3 points)

Issue

Write a program that prompts a user to enter a Social Security Number (SSN) in the following format: YYMMDD-XXXX, e.g., 820105-1234. The hyphen (-) between two parts of the SSN is optional. Thus, the SSN may consist of 10 or 11 characters. If the SSN does not meet the requirements, an error message should be printed and the user should be given the opportunity to enter the social security number again.

Based on the below described rule, the program should be able to determine and print whether the SSN belongs to a man or a woman.

Rule: If the second last digit in a social security number is even, the SSN belongs to a woman, otherwise, it belongs to a man.

The program should be able to test multiple numbers without restarting.

Test case

Please enter a Social Security Number: 8201051234

The SSN 8201051234 belongs to a man

Want to check another Social Security Number (Y/N)? Y

Please enter a Social Security Number: 198201051204

Invalid number, please enter the number in format: YYMMDD-XXXX: 820105-1204

The SSN 820105-1204 belongs to a woman

Want to check another Social Security Number (Y/N)? N

Tips

Use **cin.getline()** to read the Social Security Number. Also study the standard library **cctype** for suitable functions for character handling.

Presentation

The task is to be solved as follows:

- A fully working program that meets the specified criteria.
- A program code that is well-written and easy to read, according to the coding standards published on the course home page.

Task 5: Palindrome (3 points)

Issue

A palindrome is a word or a phrase that may be read the same way in either directions. (White space characters are ignored. Upper case and lower case characters should be equally considered – i.e. not case sensitive.) Few examples of palindromes are:

Redder
Madam
Level
Kayak
R
Never odd or even
Rats live on no evil star

Write a program that reads a word or phrase from the user, examines the word/phrase and prints if it is a palindrome or not. The program should be able to do multiple tests without restarting.

Test case

Example Run 1:

Enter a word or phrase: Redder
It is a palindorome.

Example Run 2:

Enter a word or phrase: C++
It is not a palindorome.

Tips

Read a text line into a C-string (i.e. a char array). Step through the string backwards and move the characters into a new string. Then compare the old string with the new string.

Use `cin.getline(string,dimlength);` to read the strings!

Another option is to compare the first character with the last, then the second character with the second last, etc., until you get to the middle of the word or phrase. White space characters are ignored. The first deviation will tell you it's not a palindrome and the test can be interrupted.

You can use appropriate standard functions for c-strings.

It is appropriate to create a function: `bool isPalindrome(char s[])` which performs the tests on the supplied string s.

Presentation

The task is to be solved as follows:

- A fully working program that meets the specified criteria.
- A program code that is well-written and easy to read, according to the coding standards published on the course home page.

Task 6: Grades Structure (3 points)

Issue

Write a grading program for a class with the following exam activities and grading rules:

a) There are two quizzes, each graded on the basis of 10 points.
b) There is one midterm exam and one final exam, each graded on the basis of 100 points.
c) The final exam counts for 50%, the midterm counts for 25%, and the two quizzes together count for 25% of the **total numeric score** of a student in the course. (You should take average of the two quizzes and normalize the average by multiplying with 10 before counting the average quizzes score for a 25% to the total numeric score.
d) The total numeric score is then used to determine **grade** letter. The grade letter is A for a total numeric score of 90 or more, B for total numeric score of 80 or more (but less than 90), C for the score of 70 or more (but less than 80), D for the score of 60 or more (but less than 70), and F for the total numeric score of below 60.

Write a program that asks a user to enter quizzes and exams scores for **three** students as shown in the example run below. After reading quizzes and exams' scores, the program displays each student's record, which consists of two quizzes and two exams scores as well as a student's average numeric score for the entire course and a final grade. **You are required to define and use your own *structure* type to store and output each element (quizzes, exams, average numeric score, and grade) of a student record.**

Test case

```
enter the student number: 1
enter two 10 point quizes
7 10
enter the midterm and final exam grades. These are 100 point tests
90 95

enter the student number: 2
enter two 10 point quizes
9 8
enter the midterm and final exam grades. These are 100 point tests
90 80

enter the student number: 3
enter two 10 point quizes
7 8
enter the midterm and final exam grades. These are 100 point tests
70 80

The record for student number: 1
The quiz grades are: 7 10
The midterm and exam grades are: 90 95
The total numeric score is: 91.25
and the letter grade assigned is A

The record for student number: 2
The quiz grades are: 9 8
The midterm and exam grades are: 90 80
The total numeric score is: 83.75
and the letter grade assigned is B
```

```
The record for student number: 3
The quiz grades are: 7 8
The midterm and exam grades are: 70 80
The total numeric score is: 76.25
and the letter grade assigned is C
```

Presentation

The task is to be solved as follows:

- A fully working program that meets the specified criteria.
- A program code that is well-written and easy to read, according to the coding standards published on the course home page.